Technical Specification for MapIT! (DRAFT)

CERES Development Team March 2004

At CERES, we wish to provide a general purpose spatial capture application for use by various state agencies, that enables their users to interactively define spatial footprints of interest to them in their applications. We envision this new mapping application, which we have dubbed MapIT!, to be designed modularly and extensibly, as to be applicable to projects that have a need for capturing spatial information.

In the following document, we will first explain our expectations for the MapIT! interface. An overview of the architecture follows, then we will discuss the general workings of the user interface. This will be followed by a suggested parameter passing protocol for communicating between MapIT! and external initiating applications. The potential role of sanity checks in MapIT! is described next. We then provide some implementation requirements that we would like you to adhere to in designing MapIT!, followed by a suggested list of technologies and specifications that you might consider in developing MapIT!. This paper concludes with a section on future directions, i.e. the functionality we would like to have in the next iteration of the MapIT! interface.

Note: Since this is a preliminary draft specification, none of the information presented here, especially the look and feel of the user interface, is set in stone. This document serves only to convey to you our thoughts and concerns on MapIT!, regarding its potential behavior and implementation. Feel free to let us know of any corrections or concerns to any of the items presented in this document.

Contents

1 Expectations

As we are all aware, most web based database applications are only adept at managing traditional tabular data, but very few applications are designed to capture and maintain spatial data. That is, although most applications can render and query for feature layers in their mapping applications, few applications actually enable users to interactively define meaningful spatial footprints of interest to users. For this reason, we see a great opportunity for introducing an architecture that does so.

We propose a new spatial capture application, which we have dubbed MapIT!, that allows users to interactively define geometries within its interface. MapIT! is launched from various initiating applications, such as the Prop 40/50 data entry interface or the CERES Metadata Catalog, that allows users to further define a spatial footprint for a record in the initiating application. Essentially, MapIT! functions as a helper application: it retrieves information from the initiating application (the referring host, the corresponding record in the initating application, etc.), processes the geometries as defined by the user, then passes back the information to the initiating application.

One highlight of MapIT! is its ability to define a collection of heterogeneous geometric objects associated with a given record. For instance, a user might define the extents of a hospital by the boundaries of the land where the hospital is situated on (modeled as a polygon), along with the two buildings that comprise the hospital (modeled as two points). In this case, the record is spatially represented as a *collection* of three geometric objects: two points and one polygon. MapIT! should allow users to individually update/delete individual components of a given spatial footprint (i.e. if the user wishes to delete a point in the collection, the user interface should be constructed in such a way as to make this possible).

Along the same lines, we will also provide users with the ability to reuse geometries from existing thematic layers in the definition of their geometries. For example, if the user wishes to add the extents of a specific county as a component in their defined geometry, then MapIT! will provide an interface that allows them to do so.

We also suggest that MapIT! be designed as a stateless application, at least for the first pass, to simplify its overall implementation. As a consequence of this, MapIT! will not need to be concerned at all about user authentication (except for communicating via HTTPS between MapIT! and the initiating application); that responsibility is delegated to the initiating application. Also, MapIT! is not required to persistently store the geometries. If it is necessary to store the geometries it creates in a temporary store (for rendering purposes, for example), then MapIT! may do so, but this is not mandatory. We believe that designing MapIT! as a stateless application will ensure that MapIT! is designed as flexibly as can be, so that it can be applied to the greatest number of applications.

Since MapIT! will be invoked by a wide variety of initiating applications, we would like to customize the look and feel of MapIT!, so that it matches the theme of the invoking application. For example, if MapIT! was invoked from the Department of Parks and Recreation (DPR, or "Parks") data entry interface, when MapIT! is invoked, we would want it to change its look and feel so that it matches that of Parks. Likewise, if MapIT! was invoked from the CERES Metadata catalog, this new invocation of MapIT! will change its look and feel so that it more resembles CERES. MapIT! can determine this via the parameter passing protocol, defined in the later section.

One of the potential core functions of MapIT! is its ability to run sanity checks of user defined geometries against those in the gazetteer. We at CERES are still actively debating the extent of sanity checks that can be performed. An entire section later in this document is devoted to the potential possibilities and difficulties involved with implementing sanity checks in MapIT!.

2 Architecture

As indicated in the previous section, the architecture logically consists of two components: the initiating application, and MapIT! itself. Since MapIT! is designed as a stateless application (i.e. the user-defined geometries are removed once processing is complete), MapIT! does not need to concern itself with authentication. It will however, need to communicate with the initiating application via HTTPS, to ensure that information passed between the two cannot be spoofed; this is the only authentication issue that needs to be taken care of.

Processing essentially occurs in three phases. In the first phase, the initiating application will pass parameters to the MapIT! application, to identify the application, on whose behalf MapIT! will generate spatial information for. The definition of these parameters is defined in the section called "Parameter Passing Protocol". In the second phase, MapIT! interactively processes the geometries as defined by the user. In the final phase, MapIT! will pass the generated information back to the initiating application.

To fulfill the need of reusing existing geometries in existing thematic layers such as populated places and county boundaries, it will be necessary for MapIT! to store this information in a geometry database of some sort, such as PostGIS (a spatial extension package for the open-source

PostgreSQL database). This will allow MapIT! to select specific features from the database, for inclusion in the user interface.

If needed in your implementation, you may choose to store the user defined geometries in the geometry database as well, although as mentioned earlier, you are not required to store that information persistently (that is, the geometries are ephemeral).

Figure 1 illustrates the relationship between the initiating application and MapIT!.

At a higher level, figure 2 illustrates how MapIT! interacts with various subsystems, specifically the Prop 40/50 interface and the CEIC Catalog. A detailed description of the various subsystems and their interaction is provided next.

3 System Interaction

This section describes in detail how MapIT! interacts with the various subsystems, as illustrated in figure 2. In the following, each of the referenced subsystems is in curly braces {}.

1. In the beginning, there is the Prop 40/50 site. To be specific, there are actually three sites: one for retrieving data (password protected), from Parks (http://propdata.parks.ca.gov), one for editing the tabular data (which they do not provide access to, only existing in screenshots), and one for general public browsing, under the Resources banner (http://4050bonds.resources.ca.gov).

   We are primarily concerned with the data retrieval site, and the only interaction we have with it is to download the data they have, which is performed by component {3}.

2. To not have our hands tied to the Parks interface providing hooks to MapIT!, we will have our own tabular editing interface, that allows us to browse and edit the tabular Bonds data downloaded from them. Functionally, it will be similar to the editing interface in the screenshots they provide us, except that it will have a button to launch the MapIT! interface.

   The interface is semi-public, that will be accessible via privileged users. In the Implementation Requirements section, two authentication strategies are presented (i.e. via database, or via CaSIL LDAP), that can be used to allow authorized users to edit their data.

   One outstanding question is: does the tabular interface process the results returned from MapIT!; i.e., the user-defined geometric extents for the record in question? If so, does it then take the information from MapIT!, parse it in some way, and then update the entries in the spatial data storage component? Now that we do have this interface, the answer is now a definite yes. That being the case, there will be a GML parsing component to this application.

3. In order for the CERES interface {2} to function, a download manager module will need to download the dataset from Parks, and convert that data to an ESRI Shapefile.

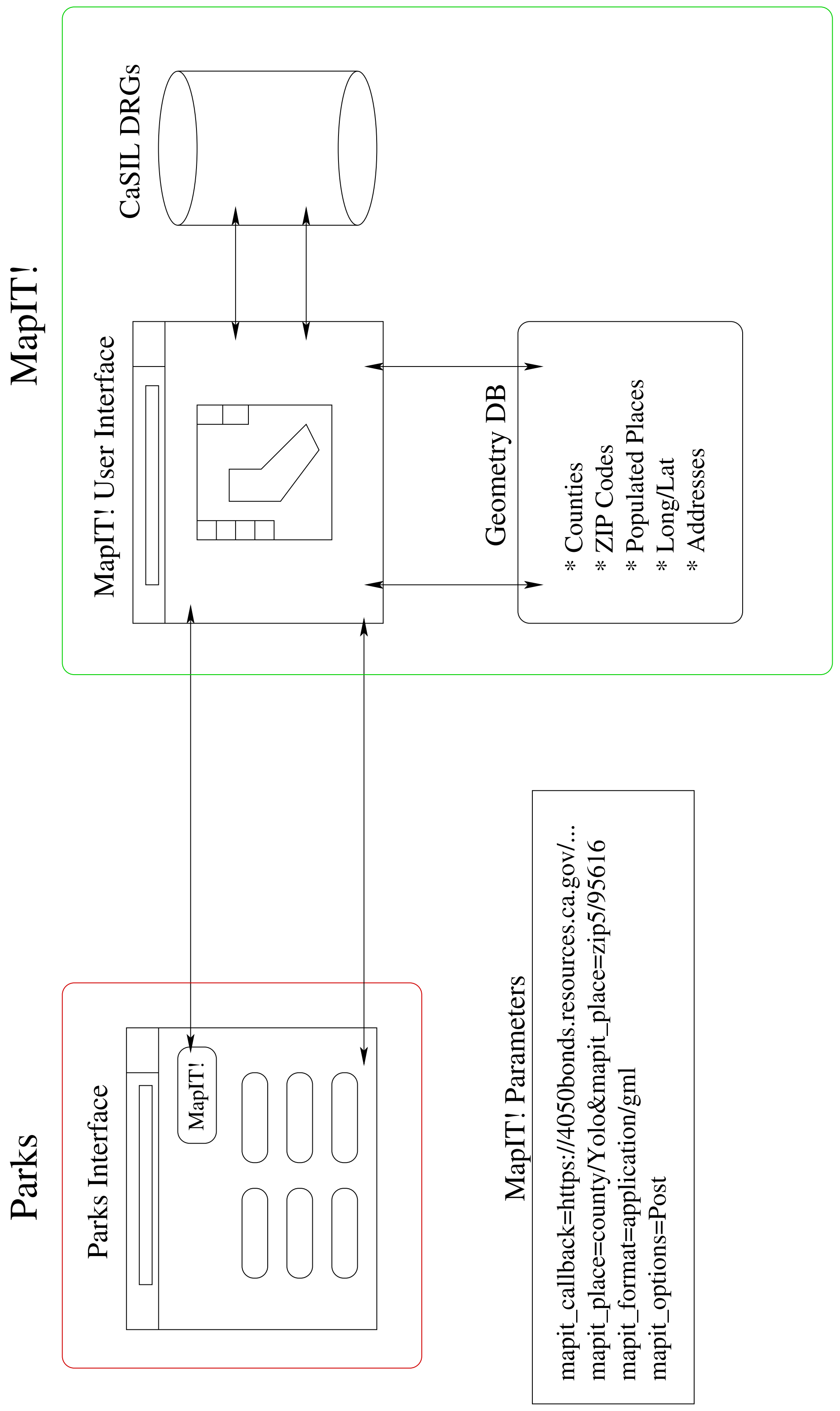


Figure 1: MapIT! Architecture

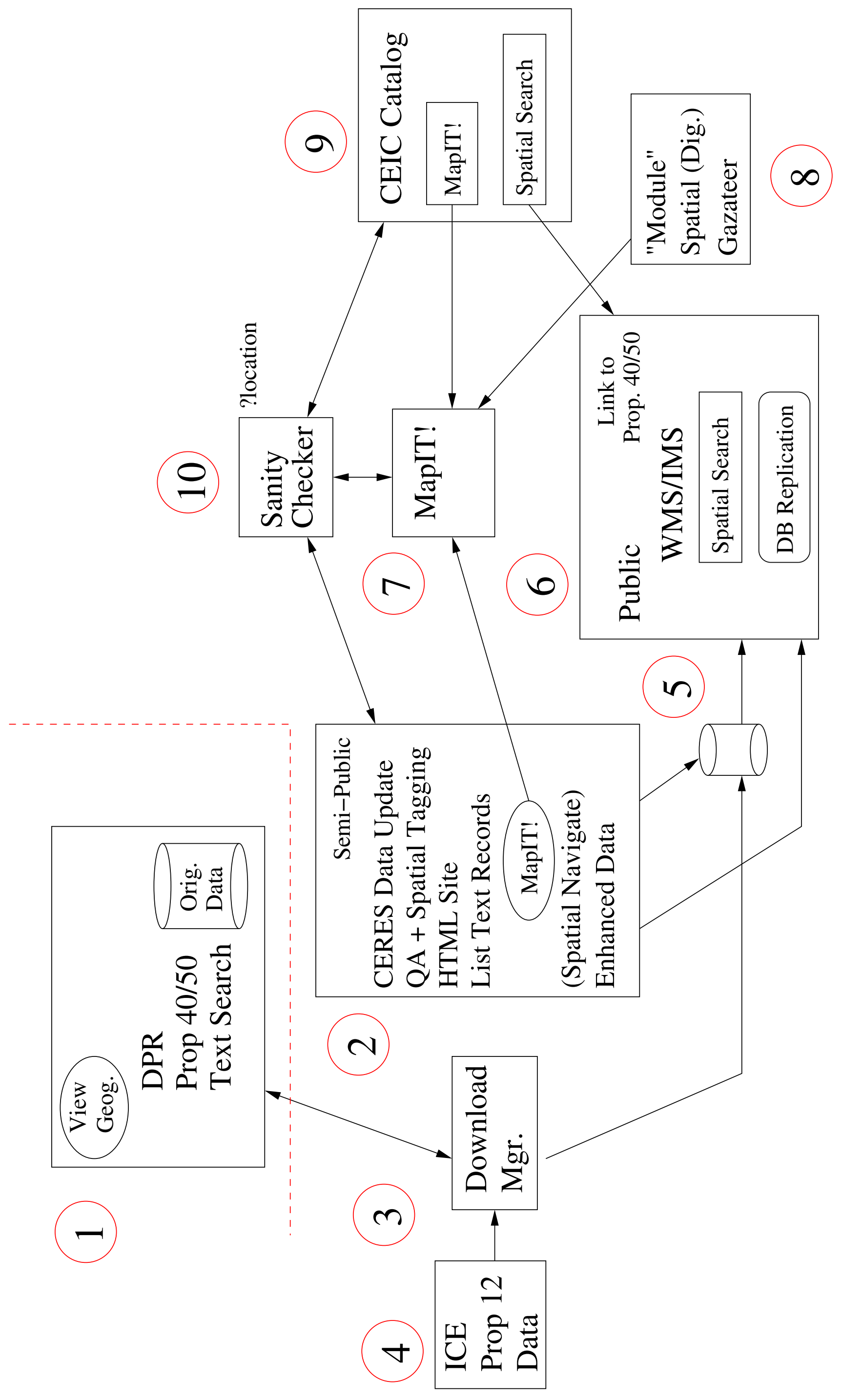


Figure 2: MapIT! System Interaction

Note the following caveats:

- The data retrieval page at Parks requires that the user download from Internet Explorer ONLY.
- Worse, previously they had firewall rules (or some other blocking mechanism) that ONLY allows external access at CERES. There wasn't a way that we can access it at UCD or for the GISC folks at UCB to access it, unless Parks is willing to change their setup. Fortunately, that restriction seems to be lifted recently.
- The comma-separated values (CSV) dump needs to be sanitized for proper importing in PostGIS. One needs to first import the data in MS Access, and convert it to tab-separated format. A ”tidy” script then needs to clean up the output tab-separated format data before one can import into PostGIS; depending on the data, it's actually a lot of work dealing with the exceptional conditions.

For these reasons, we at CERES have taken care of all these steps; i.e. download the data, and convert to shapefile. The Implementation Requirements section describes where you can download the ESRI Shapefiles based on entries in the Prop 40 database.

4. Besides getting data from Parks, the download manager {3} can access other datasets as well (ICE Prop 12 data), and store it.

5. This spatial data storage component stores the representation of the Prop 40/50 data. If using PostGIS, it allows you to generate an ESRI Shapefile, based on the current contents of the database. It will also have the five (or more) feature layers as discussed previously.

6. One point of potential controversy is that there will be two interfaces to the spatial data: one for browsing of the spatial data by the general public, discussed in this section (similar to those provided by ArcIMS and Mapserver), and the other one being the spatial footprint editing application; i.e. MapIT! ({7}, below). The policy would be, for users to edit, they must go through the CERES-Parks editing interface {2}.

   Now, to browse the data, if currentness is not an issue, this browsing module can get away with accessing the spatial data in the ESRI Shapefile. This is the case for ArcIMS; if I understand correctly, the only way for it to access dynamically changing feature data is for it to use ArcSDE, which is not an option for us. Of course, the data may not be the most current if the database entries changes, so periodic updates will be necessary.

   Otherwise, the Mapserver/PostGIS combination is the best course of action here.

7. The *raison d'être*, of the whole architecture, is of course, the MapIT! spatial data editing application. It is launched from various initiating clients (i.e. the CERES Parks editing

interface, {2} and the CERES catalog, {9}). If specified by the client, it will return the generated data, usually in GML format, back to the initating clients, for further processing.

8. A Spatial Gazetteer is needed, that will store the exact representation for place names (and other features as well).

9. The CEIC Catalog is the other major application that will leverage MapIT!. Again, it will launch MapIT! via a button in its data entry form. It will also perform spatial searches by launching the browse application {6}.

10. The Sanity Checker. Given its importance, a later section in this document is devoted to describing the role of sanity checking in MapIT!

4 User Interface

This section discusses the various components in MapIT!'s user interface, according to function.

A mockup page for the user interface, which will be referred to throughout this section, is located at:

http://sample1.casil.ucdavis.edu/submission/edit.html

4.1 Tool Information

Common to the following two subsections is the tool information area, which provides interactive information about the currently selected tool (zoom to initial extents, add line, etc.), and how best to utilize the tool. This section is illustrated in figure 3.

4.2 Browsing

The browsing tools are similar to tools used in many conventional mapping interfaces, such as ArcIMS and Mapserver clients. In the mockup page, there are six browsing tools on the left hand side of the image. These tools are illustrated in figure 4.

The first two buttons are used for zooming in and zooming out, either by clicking on the map, or by dragging a rectangle that is used to calculate the new map extents. The third button is used to recenter the map image by clicking on the desired center point on the map. The fourth button is used to zoom to initial extents, which will be the State of California in our implementation. The fifth tool allows you to zoom to the extents of the current feature. The sixth tool allows you to select individual geometries on the map.

Elsewhere in the map interface, there is a keymap that allows users to quickly zoom to other parts of California, as illustrated in figure 5.

An Image Size box allows users to change the size of the image, as illustrated in figure 6.

Finally, the scale bar allows users to quickly change from one scale factor to another, as illustrated in figure 7.

A novel feature of MapIT! is the thematic navigation section, that allows users to zoom to specific features found in thematic layers. This section is illustrated figure 8.

In the mockup page, there are five thematic layers users can use to navigate with:

- County - County layer (FRAP)
- Zip Code - Zip code from census data
- Place Name - Populated Place Names (Teale)
- Long/Lat - a long/lat value
- Address - address information from census data

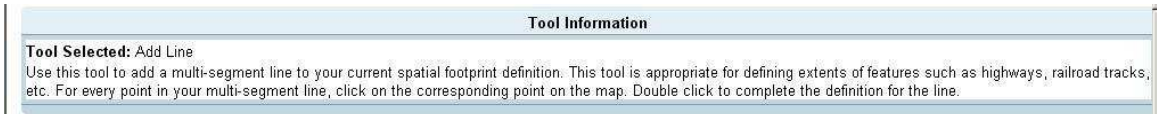
So for instance, if a user wishes to zoom to the extents of San Mateo county, she just needs to select San Mateo from the drop down list, and MapIT! will automatically zoom to the extents of San Mateo county.

In most cases, the value as entered by user is potentially ambiguous (i.e. does the user wish to zoom to the City of Davis, or to the Davis Reservoir?). So once the user enters her value into the input box, a pop-up window will show a table, listing the various features that match the user's entered value. The user will then choose one of the features from the pop-up window, at which point MapIT! will zoom to the extents of the chosen feature.

4.3 Editing

The three editing tools allow users to create a set of geometries that is used to represent the spatial footprint of the user-defined feature. These tools are illustrated in figure 9.

The functionality of these tools are described as follows.


Tool Information

Tool Selected: Add Line

Use this tool to add a multi-segment line to your current spatial footprint definition. This tool is appropriate for defining extents of features such as highways, railroad tracks, etc. For every point in your multi-segment line, click on the corresponding point on the map. Double click to complete the definition for the line.

Figure 3: Tool Information

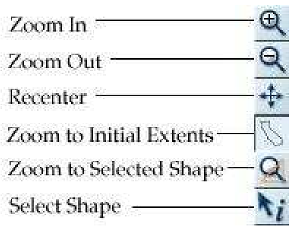


Figure 4: MapIT! Browsing Tools

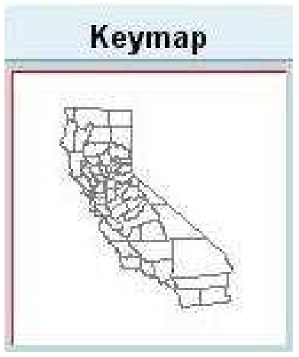


Figure 5: Keymap

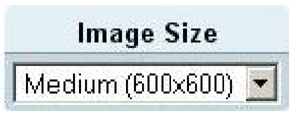


Image Size

Medium (600x600)

Figure 6: Image Size

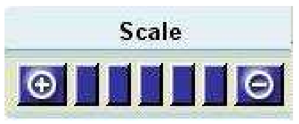


Figure 7: Scale Navigation

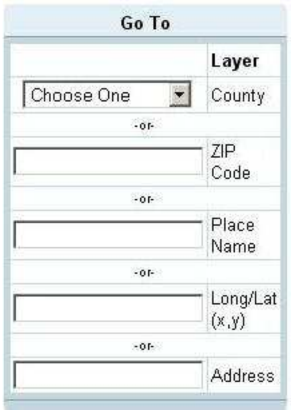


Figure 8: Thematic Navigation

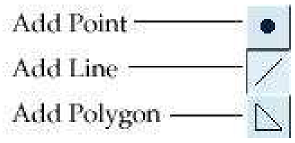


Figure 9: MapIT! Editing Tools

4.3.1 Shape Creation

In the mockup page, there are three editing tools on the right hand side of the image. These tools are used to draw point, line, and polygon primitives on the map.

The point tool allows the user to click on the map to render the point.

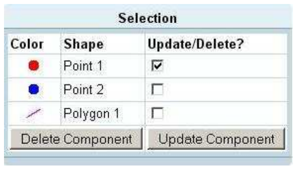
The line tool allows the user to create a multi segment line, by clicking on the multiple vertices of the multi segment line; double clicking on the last point finishes the definition of the line.

The polygon tool behaves similarly to the line tool, in that users can click on the points of the polygon. When the user double clicks on the final point, MapIT! will complete the polygon definition, by drawing a segment between the first and last points.

4.3.2 Geometry Set Display

As mentioned earlier, MapIT! allows users to define their spatial footprints as a collection of heterogeneous geometric objects. To help users make sense of what geometries are currently in their geometry set, a section on the web page is devoted to indicating what objects are currently in the set, and allows the user to delete or update the individual geometry objects.

Towards the bottom of the mockup page, the Selection table lists the various geometries currently in the geometry set. Each table row indicates the type of geometry, the color the geometry appears on the map, and a check box, that allows users to select the geometry object for deletion or update. We suggest color coding the various geometries, to better help users identify the various geometries in their current selection set. This is illustrated in figure 10.



Selection

Color	Shape	Update/Delete?
●	Point 1	☑
●	Point 2	☐
⟋	Polygon 1	☐

Figure 10: Geometry Set Display

To modify individual geometry components, the user needs to select the component in the geometry set display, and choose either the delete button or the update button.

Pressing the Delete Component button allows the user to delete the geometry components that have check boxes next to them.

Only individual geometry components are subject to update. To perform a geometry component update, the user first clicks on the Update Component button, with exactly one of the geometry components having a check box next to them. MapIT! will then hide the selection region, and prompt the user to redraw the geometry component using one of the three shape creation tools. For reference, the old geometry should be shown in a different color (perhaps shaded) to give the user a bearing as to what the original geometry component looks like, to better help them define the new geometry. Based on the shape creation tool, once the shape definition is complete, the spatial footprint is updated with the newly defined geometry component.

4.3.3 Add From Existing Geometries

Users can add new geometry objects, based on existing features from the five navigation layers listed earlier. For instance, if the user wishes to add the extents of Yolo County to the current geometry set, she would just select Yolo county from the drop down list, and it would be added to the geometry set. This is illustrated in figure 11.

As in the case for the thematic navigation section, if the selection is ambiguous, a pop-up window would be displayed, allowing the user to choose one of the available candidates. Upon selection, the new geometry will be added to the current selection set.

4.4 Geometry Annotation

In the “Add Description” box, the user can enter an annotation to the currently defined spatial footprint, to better identify or clarify what the spatial footprint actually represents. This is illustrated

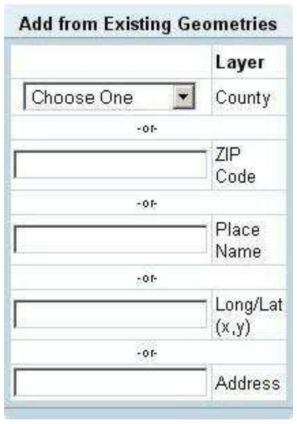


Figure 11: Add From Existing Geometries

in figure 12.

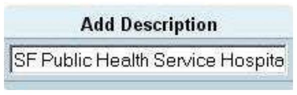


Figure 12: Add Description

4.5 Result Format

This pull down box specifies the type of result format that is sent from MapIT! back to the initiating application. This information is specified as part of the parameter passing protocol, as described below. When multiple result formats are passed to MapIT!, this menu allows the user to choose which of the listed result formats should be passed back to the initiating application. As described in the next section, the one mandatory result format will be GML. This is illustrated in figure 13.

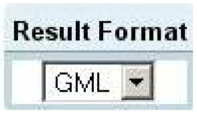


Figure 13: Result Format

4.6 Data Download

Here, a user can choose to download the currently defined spatial extents in one of two formats.

The first download format is the customary ESRI Shapefile format. Since a user can define heterogeneous geometries, a zipped file containing multiple shapefiles (one per shape type) is sent to the user. In essence, this provides a sort of clip and ship capability of the user defined geometry.

The second download format is GML. Besides sending a copy of the GML data to the initiating application (typically specified in the result format area), the user can directly download the GML data, perhaps for use in their own GML-based applications.

The data download section is illustrated in figure 14.

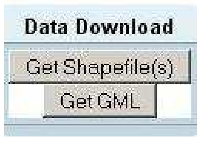


Figure 14: Data Download

4.6.1 Commit/Revert Footprint

Due to the need of maintaining a geometry collection, the selection set may undergo many changes before the user arrives at a spatial footprint she is finally satisfied with. If and when the user is satisfied with her footprint definition, she can click on the "Commit Footprint" button to have MapIT! send information about the rendered geometry, back to the initiating application. Conversely, if the user wishes to start all over in her footprint definition, she can click on the "Revert Footprint" button to restore MapIT! to the original state when the user launched the application. This is illustrated in figure 15.

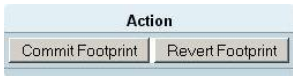


Figure 15: Commit/Revert Footprint

5 Parameter Passing Protocol

A standard protocol needs to be defined for communicating between initiating applications and MapIT! Here we propose a general parameter passing protocol, that allows initiating applications to leverage MapIT! in capturing user-defined spatial extents.

5.1 Protocol

We propose that all initiating applications send the following four parameters to MapIT!, in order for MapIT! to properly send back meaningful information to the application.

5.1.1 mapit_callback

This parameter specifies the referring host, on whose behalf MapIT! is invoked to render the user-defined geometries. For example,

mapit_callback=https://ceic.gis.ca.gov/parse_geom.cgi?record_id=4267&enter_place&session_id=499770&

Here, you see that the protocol is HTTPS, and that a session_id is provided to MapIT!, so that MapIT! can return the information properly for the given record.

For interface rendering, MapIT! might have an internal list of domain names, which it could match to the one in mapit_callback, to customize the display of the interface (i.e. generate a Park's version of MapIT!, vs. a CEIC version of MapIT!, from the point of view of the user).

5.1.2 mapit_place

This parameter specifies a list of various thematic layers, that will be used to zoom MapIT! to the approximate location of the user defined geometries. For example,

mapit_place=county/Yolo&mapit_place=zip5/95616

5.1.3 mapit_format

This parameter specifies the various output formats that the initating application can process properly. For example,

mapit_format=application/gml&

One format that is proposed is application/gml: GML is OpenGIS's format for standardizing the representation of geometries in XML documents. It should be noted that GML isn't nearly as widely deployed; once GML processors are widely available, GML provides the portability necessary to convert between GML and other data formats, such as ESRI Shapefiles.

The technical specification for GML can be found at:

http://www.opengis.org/docs/02-023r4.pdf

Outstanding issue: what other formats (besides GML, which is essentially XML) would initiating applications want?

As mentioned in the user interface section, there will be a drop down box, that lets the user choose which of the various return formats the initiating application should get back from MapIT!.

5.1.4 mapit_options

This parameter specifies an extensible list of options to provide additional information regarding the initiating application. For example,

mapit_options=edit&mapit_options=post&

Two options we intend to implement are:

a. Edit/View: determine the mode of operation for MapIT!: should it be in view-only mode, or should it be in editing mode. Under normal circumstances, MapIT! will be in the editing mode.

b. Get/Post: an HTTP issue, specify the form protocol used to submit data from MapIT! back to the initiating application (in this mode, the initiating application is most likely a CGI application that retrieves the processed information by means of form parameters).

’Get’ mode encodes form parameters in the URL; Roger notes that there is a practical/implementation limit in the amount of information you can pack into the URL, depending on the browser. ’Post’ mode, on the other hand, stores form parameters in the HTTP header. As a general rule, ’Post’ is the desired method for returning information back to the initating application.

5.2 Result Format

As indicated earlier, one of the result formats is application/gml. Besides being an OpenGIS standard, the advantage of returning the MapIT! results in GML is mainly an issue of portability. As more and more GML processors become commonplace, the GML format readily allows conversions between it and other formats, such as ESRI Shapefiles.

Feel free to suggest and/or implement result formats that you feel initiating applications would find use for.

6 Sanity Checking

One potentially significant benefit of MapIT! is the ability to run a battery of sanity checks on the user defined geometries, to determine if those definitions are consistent with the features as specified by the initiating application. Among the CERES staff, the consensus is that, even though it is not an all-encompassing solution, there are a limited set of checks that could be performed via the sanity checker.

For instance, if the initiating application specifies a place name of Yolo County, with a zip code of 95616, and the user defines a geometry does not fall within the extents of Yolo County, or the zip code region, then the sanity checks can flag the inconsistent geometries. The next time the user logs in, a pop up window can be presented that lists all inconsistent geometries, and allow the users to rectified any invalid geometry definitions. The exact geometry definitions would be the entries in a canonical gazetteer.

Here at CERES, we have a crucial need for validating applications to provide some level of consistency in all our captured spatial data, as evidenced by our ongoing link checker application. A general purpose sanity checker, coupled to MapIT!, would be tremendously beneficial for us in validating across various geometries defined regardless of the initiating application.

On one hand, we can validate to a certain degree, by having entries in the database that are tagged as to whether the entry is valid with respect to a certain feature layer. For example, a column in the database table that represents user-defined geometries could indicate whether the user-defined footprint validates against the footprint of the counties / populated place names / zip code (etc.) layer. (This of course assumes the features in the various navigation layers are correct and deemed exact. It also assumes that the user-defined geometries are actually stored persistently).

On the other hand, in the case of the CEIC catalog, all bounding box values in the CEIC Catalog will be updated to corresponding values in the Gazetteer {8} once that component is in place. Once that's done, there really isn't a sanity checking component to this when that's completed, since the geometries are exact. To elaborate, for the catalog entries, you could select a geometry in the gazetteer (from the "Add to Existing Geometries" section) that will take that spatial footprint, and deem it as correct (since the Gazetteer entries are treated as exact definitions). If the user chooses the Gazetteer entry as the EXACT footprint, how can we definitely say that the user was in error for the definitions that she just made?

The issue essentially boils down to the following constraint between the initiating application and MapIT!. Recall that parameters passed from the initiating application to MapIT! tells it where to zoom to at the start of the session. For instance, the CERES Parks interface has a ZIP code field, which if filled out, is then passed to MapIT!. The "Go To" section in MapIT!, under the ZIP Code section, will then have that value filled in initially, and the map is recentered to the specified ZIP Code.

Now, if the user zooms to a different part of California and defines her spatial footprint, using an existing geometry from the Gazetteer, that is outside of the specified zip code, would that AUTOMATICALLY qualify as an error?

If so, the passed parameter serves, not only a navigational role, to initially zoom the user to the specified location, but also as a CONSTRAINT, that mandates that all user-defined geometries MUST fall within the extents corresponding to the passed parameter.

Note that the Gazetteer footprint definition cannot be in error, since it is already exact, but if the above constraint is maintained, then using the exact Gazetteer footprint would still be in error, because of the CONTEXTUAL information passed by the initating application.

Given the importance of sanity checking in MapIT!, we would like you to indicate to us what aspects of sanity checking are implementable within the time allotted in our contract, and how these sanity checks serve to help users in defining consistent geometric footprints.

7 Implementation Requirements

The following are implementation requirements we ask of you when designing MapIT!

7.1 Use of Freely Available Software

We ask that, in implementing MapIT!, that the technologies utilized be as close to being completely based on freely available software as possible. We ask this because we will maintain MapIT! for many years; given the current budget climate, we would like to cap costs as best we can, and would try avoid entering into expensive licensing agreements for the use of certain technologies.

7.2 Complete Access to Source Code

Along the same vein as using freely available software, we ask that we have complete access to the source code you have written in creating MapIT!, annotated with documentation comments whenever and wherever appropriate.

7.3 Browsing Portions WMS compliant

We ask that the browsing portions of MapIT! (i.e. the zoom and pan features) be completely OpenGIS Web Map Services (WMS) compliant. In other words, we ask that MapIT! be a WMS server, so that WMS clients can connect to MapIT! to discover what layers are available via the WMS GetCapabilities request, and render maps on-the-fly using the WMS GetMap requests.

7.4 Only CaSIL Data in MapIT!

To standardize on the kinds of data used in MapIT!, we ask that only CaSIL data be utilized in MapIT!.

CaSIL Digital Raster Graphics (DRGs) shall be the only background layer available to users when they browse via MapIT!. We believe the topology data from the DRGs is more than sufficient in helping users determine what region of California they are looking in, that incorporating any additional layers tend to confuse our users more than to help them.

The CaSIL DRGs come in 1:250K (C-series), 1:100K (F-series), and 1:24K (O-series). As such, we don't have any maps above 250K scale; we would like your input as to what maps are appropriate above this scale, and we will obtain these maps for inclusion in MapIT!.

7.5 Standard Scale for Representing Geometries

The issue of precision and accuracy, as you have suggested in our initial meeting, is a tricky issue to deal with in any mapping application. To tackle this issue, we suggest that you represent user-defined geometries in MapIT! at 1:20 million scale. By standardizing on a common scale, it is then possible to meaningfully compare the extents of user-defined geometries.

7.6 Authentication

Once you have completed the stateless implementation of MapIT!, and would like to move towards converting the stateless implementation to a stateful implementation (for capabilities such as the automated auditing of the user-defined geometries), we ask that you investigate two different authentication strategies.

The first authentication strategy is to authenticate against user credential information in a MySQL database. Specifically, you will be authenticating against the username/password information stored in the CEIC catalog. We intend to provide you with a wrapper script to allow MapIT! to transparently validate user credential information against the MySQL database..

The second authentication strategy is to leverage GForge's existing LDAP authentication on CaSIL, and design MapIT! to interface with CaSIL GForge's LDAP server. The reason for this is we would like initiating applications to standardize on a common authentication mechanism; on CaSIL, we have enabled LDAP Transport Layer Security (TLS) to ensure that the exchange of credential information via LDAP be securely encrypted.

7.7 Provided Datasets to be placed in GForge

We ask that all provided datasets be placed in GForge's data download area. That is, an ESRI Shapefile that shows the locations of the Prop 40 database be placed in CaSIL GForge.

We at CERES have performed this step. You can access these datasets (starting on March 1, and updated on a weekly basis), at:

http://casil.ucdavis.edu/project/showfiles.php?group_id=29

8 Suggested Technologies

We suggest using the following technologies when developing the MapIT! application. You are of course free to develop your our software for this purpose, as long as they adhere to the technical guidelines that were indicated earlier.

8.1 PostGIS

PostGIS is a software library package that enables the representation of geographic objects in the open-source PostgreSQL database. With PostGIS, you can represent geographic objects based on the OpenGIS Simple Features for SQL specification. Geometries can easily be reprojected, and ESRI Shapefiles can readily be generated/imported using a command line interface to the PostGIS database. Finally, when compiled against the GEOS topological library, you can apply topological functions, such as convex hull, to your geometries.

Information about PostGIS can be found at

http://postgis.refractions.net

Information about the GEOS topology library can be found at

http://geos.refractions.net

The OpenGIS Simple Features for SQL specification can be retrieved at

http://www.opengis.org/docs/99-049.pdf

8.2 UMN Mapserver

The University of Minnesota (UMN) Mapserver is an open-source map server application that can quickly render individual raster and vector layers. The latest version has support for rendering true-color images, and the ability to define no value regions, useful for masking out undefined regions in an image.

A PostGIS driver included in the package allows you to display the live contents from PostGIS, allowing you to easily render changes to the geometric representation of your geometries.

Mapserver also provides a direct API to map layers via its Mapscript API. APIs are provided for the Perl, PHP, and Java languages.

Information about UMN Mapserver can be found at:

http://mapserver.gis.umn.edu

8.3 MapLab/Rosa Applet

The MapLab package allows you to quickly build prototype Mapserver applications. The first module allows you to easily create configuration files using a forms based interface, the second module allows you to view the new service created using Web Map Service calls, and the third module allows you to create custom standalone map interfaces.

The Rosa Applet allows you to draw simple geometric objects on an image. The applet allows you to place tool buttons on the map image, which sets the specific flag when a user selects one of the tools. Map applications based on Mapscript can then perform the appropriate actions based on the flags set by the tools.

The MapLab package can be found at:

http://www.dmsolutions.ca/techserv/maplab.html

Information about the Rosa applet (which is also included in MapLab, and available as a separate download), can be found at:

http://www2.dmsolutions.ca/webtools/rosa/

A sample online application that uses Rosa and PHP Mapscript can be found at:

http://www2.dmsolutions.on.ca/gmap/gmap75.phtml

The source code for the above applications can be downloaded at:

http://www2.dmsolutions.ca/mapserver/dl/

9 Future Directions

Once the stateless version of MapIT! is completed, we might later choose to extend MapIT! so that it does persistently capture the user-defined geometries. This implies that access control/authentication will now be needed. That is, to prevent unauthorized users to view and/or modify the spatial footprints of other users, MapIT! would now have to be concerned with authentication. Decisions will need to be made as to how easy it is for MapIT! to integrate with different varieties of authentication mechanisms, or whether applications are required to leverage GForge's LDAP authentication mechanism.

Furthermore, with the persistent storage of user-defined geometries, users can now browse among the various geometries they have created. Recall that in the stateless version of MapIT!, the user can only edit the geometry associated with a record from the initiating application. With the persistent storage of user-defined geometries, it is now possible for MapIT! to display all geometries that were created by the same user.

With persistent storage of user-defined geometries, for administrators of MapIT!, we can also access a comprehensive list of all invalid geometries. Administrators can manually review the list, and clear those geometries that are flagged incorrectly by the sanity checks (i.e. false positives). We can also determine what types of geometries most users are inclined to utilize in describing their geometries. For a more personalized experience, for those users with a provided e-mail address, we can mail a list of geometries directly to those users, that inform them of the inconsistent geometries.

As described in this document, MapIT!'s functions similarly to an OpenGIS Web Feature Service (WFS):

http://www.opengis.org/docs/02-058.pdf

Specifically, WFS provides compliant clients a standard way of determining the various feature types offered by the system, and a standard way of retrieving specific features based on feature attributes or spatial selection. For transaction-enabled WFS systems (optional according to the spec but highly desirable), WFS provides a standard way for features to be modified. The results of WFS requests are in GML. Time permitting, we would like to investigate whether the MapIT! system can truly be designed to be WFS compliant.